Theoretical Computer Science 410 (2009) 332–346

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Complexity classes for self-assembling flexible tiles

Nataša Jonoska *, Gregory L. McColm

Department of Mathematics, University of South Florida, Tampa, FL 33620, United States

ARTICLE INFO

Keywords:
Self-assembly
DNA junction molecules
DNA-based graph structures
Assembling complexes
Complexity classes of self-assembly

ABSTRACT

We present a theoretical model for self-assembling DNA tiles with flexible branches. We encode an instance of a "problem" as a pot of such tiles for which a "solution" is an assembled complete complex without any free sticky ends. Using the number of tiles in an assembled complex as a measure of complexity we show how NTIME classes (such as NP and NEXP) can be represented with corresponding classes of the model.

1. Introduction

One of the most striking proposals for a new kind of computer is the one based on self-assembly of molecular "tiles" made of DNA, in which the assembly process is regulated by the chemical bonding properties of DNA. Several DNA computations have been conducted, and many researchers are hopeful that this technique may be used in practice some day. Meanwhile, the empirical and theoretical experience gained by research in DNA self-assembly can be useful in researching other self-assembly phenomena, for which very little theoretical base exists.

In this paper, we present an algebraic model for a type of DNA computation (based on the self-assembly of "flexible tiles") introduced in [14–16] and [28], and we use this model to determine the computational power of this type of DNA self-assembly. It turns out that the classes of queries computed in this way correspond (in a sense) to the NTIME classes of queries.

The outline of the paper is as follows.

- In Section 2 we briefly outline the background of this type of DNA self-assembly, concentrating on the use of "flexible tiles", i.e., tiles that can change shape in order to adhere to each other with minimal geometric constraints. We outline our goals in modeling these computations.
- In Section 3 we present the basic definitions for our model as a computing system. We present a formal description of how tiles in a "pot" can assemble into "complexes".
- Section 4, contains two examples that provide an inspiration for the complexity classes defined in the following section. The first example is the question of whether a graph has a cycle, and the second is 3SAT.
- Section 6 contains the main representation theorem: given a suitable class $\mathcal{F}$ of bounds on the number of tile types, the class of queries flexible-tile assembly computable with these bounds is precisely NTIME($\mathcal{F}$).

A preliminary abstract of this work was presented in [17]; further work was presented in [19], and [20], while this model was compared with the more classical model with rigid tiles in [18].

2. Flexible branched junctions with sticky ends

DNA occurs naturally as pairs of strands of nucleotides linked by hydrogen bonds in ladder-like rungs. To join, the two strands must be (Watson–Crick) complementary: an adenine (A) nucleotide on one strand links to a thymine

* Corresponding author.
E-mail addresses: jonoska@math.usf.edu (N. Jonoska), mccolm@math.usf.edu (G.L. McColm).

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.09.054

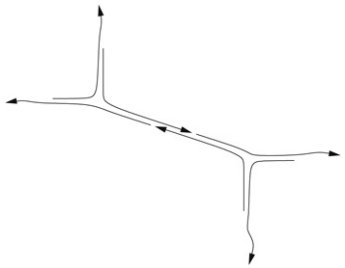

Fig. 1. Bonding of two DNA junction molecules. The double helix structure is not depicted for simplicity. The arrowhead indicates the 3′ end also ending with a single-stranded sticky end sequence. (The strands are oriented, with a 3′ or "ascending" direction, the opposite being the 5′ or "descending" direction; the two strands of a DNA molecule have opposite orientations.)

(T) nucleotide on the other, just as a guanine (G) nucleotide on one strand links to a cytosine (C) nucleotide on the other. A collection of single strands can be glued along complementary segments to form "branched junction DNA molecules" [29,34,35] which can serve as the basic building blocks in (self) assembly experiments. Given these tinkertoy-like molecules – here called *tiles* – they can be assembled into more complex structures. The tiles, i.e., branched junction DNA molecules can be designed such that each arm ends with a single stranded extension called a sticky end, or a *port*. In this sense, a regular double stranded molecule with both ends extended with sticky ends, can be seen as two-armed branched junction molecule. A sticky end joins with another sticky end containing a Watson–Crick complementary sequence (a pair of tiles joining along sticky ends is depicted in Fig. 1). In this way, two (or more) branched junction molecules can assemble in more complex structures. Starting with a test tube (called a *pot* here) of a mix containing carefully designed tiles whose ports, i.e., sticky ends, are encoded in a predefined way, more complex and graph like structures can be assembled. Such structures include more complex building blocks, such as DX and TX molecules [11,22,30], including two-dimensional arrays [36,37], three dimensional graph-like structures [14,28,38] and polyhedra [6,39,31,9].

It was also observed that the coding of the sticky ends (ports) can be used in a way that a computation, or a result of an information processing, is obtained. Starting with the first Adleman's experiment [1], several instances of computations using DNA tiles have been obtained experimentally: binary addition (simulation of XOR) using triple cross-over molecules (tiles) [23], a 9-bit instance of the "knight problem" using RNA [10], and a small instance of the maximal clique problem using plasmids [13], to name a few. Authors in [27] used algorithmic self-assembly of DNA Wang-like tiles (DX molecules) to simulate cellular automata and obtain two-dimensional arrays of the Sierpinski triangle, as well as basic computing operations of copying and counting [5]. Recently, Rothemund developed a relatively straightforward technique of designing and self-assembling arbitrary shapes [25].

However, there is a real need for understanding the limitations, boundaries, and complexity of the self-assembly process. Much of the theoretical work to describe and understand self-assembly has been concentrated on the self-assembly of Wang-like tiles, i.e., rigid DNA molecules, but only a few theoretical results have been obtained. The complexity, measured as the number of kinds of tiles needed for a unique assembly of $n \times n$ squares, is considered in [26], where it was observed that only $O(\log n)$ molecules are needed for this task. Comparison of such "shape" complexity with Kolmogorov complexity is investigated in [32]. The same model was used to theoretically observe a possible tile self-assembly of a cube [21]. In [2], computing the minimal number of tiles needed for unique self-assembly in a given shape proved to be NP-hard, and the question of whether a given set of tiles arrange in an infinite ribbon-like shape was proved to be undecidable [3].

In this paper, we consider a different model. We consider the self-assembly process of *flexible branched junction molecules*. Molecules with long arms are flexible, and those that have short arms can be made more flexible by adding a non-hybridized short sequence of T's at the junctions or along the arms. Bulged T's were used in the construction of the graphs in [14,28, 38]. This permits the construction of arbitrary graphical structures without being limited by geometric constraints.

This model was initiated in [15,16], and a heuristic model to predict the distribution of products of self-assembly was presented in [19]. In [24], by allowing a different strength on the sticky ends a model allowing disintegrating, as well as annealing, was investigated. There, the computational complexity as to whether certain pots produce a complete complex was investigated. A related follow-up work was also done in [4]. In this paper, we take a somewhat reverse course from [24]. We ask what are the problems solvable by flexible tile DNA self-assembly. Using the number of tile types as a measure of space complexity, we find that for a suitable class $\mathcal{F}$ of functions $\mathbb{N} \to \mathbb{N}$ pots that yield complexes with tiles $f \in \mathcal{F}$ correspond to the non-deterministic $\mathcal{F}$-time complete problems. Specifically, on one hand, we find that all flexible tile DNA computations are reducible to nondeterministic integer programming problems, and on the other hand, any nondeterministic computation can be simulated by a flexible tile computation. The initial introduction of the model described here was presented in [17] where the result was introduced when $\mathcal{F}$ is the class of polynomials. Here, we give a detailed theoretical description of the self-assembly model of flexible tiles and prove the correspondence of standard nondeterministic complexity classes with corresponding classes of pots yielding complexes with a bounded number of tiles.

2.1. Preliminaries and notation

The number of elements in a finite set A is denoted by $\#A$. The minimal element of the set of natural numbers $\mathbb{N}$ is 0. Let H be a finite set. A *labeled multigraph G with labels in H* is a 4-tuple $G = (V, E, \eta, \lambda)$ where V is a finite set of vertices, E is a finite set of edges, $\eta : E \to P_V$ is a function from E to the set of two-element subsets of V, the set P_V, and $\lambda : E \to H$ is a labeling function. For an edge $e \in E$ the two vertices in the set $\eta(e)$ is the set of *endpoints* of the edge e. As G is a multigraph, there might be several edges with the same set of endpoints. Note that this definition does not include multigraphs with loops. When η is injective, G is simply a graph.

For a vertex $v \in V$ we define *the neighborhood of* v as $N_v = \{ w \mid w = v \text{ or } \exists e \in E,\ \eta(e) = \{v, w\} \}$. We abuse the notation: the subgraph of G generated by N_v (vertices in N_v and all edges with end points in N_v) is also denoted with N_v. The *degree* of a vertex $v \in V$ is $deg(v) = \#\{ e \mid v \in \eta(e) \}$. A subgraph G' of G generated by the vertex subset V' is the subgraph that consists of vertices V' and all edges that have endpoints in V'. A subgraph G' is generated by an edge subset E' if it consists of all endpoints of the edges in E' and edges E'. The labeling of the edges is inherited from G.

Given two labeled multigraphs G_1 and G_2 with labels in H_1 and H_2 respectively, a *graph homomorphism* from G_1 to G_2, denoted $\phi : G_1 \to G_2$, is a triple of functions $\phi = (\phi_v, \phi_e, \phi_h)$ such that $\phi_v : V_1 \to V_2$, $\phi_e : E_1 \to E_2$ satisfying $\phi_v(\eta_1(e)) = \eta_2(\phi_e(e))$ and $\phi_h : H_1 \to H_2$ satisfying $\phi_h(\lambda_1(e)) = \lambda_2(\phi_e(e))$ for all $e \in E_1$. As is standard in graph theory, we write ϕ for any of ϕ_v, ϕ_e, or ϕ_h when it is clear from the context which one of the three is used. In this paper, we require that ϕ_h be the identity when $H_1 = H_2 = H$.

A graph homomorphism is onto if both ϕ_v and ϕ_e are onto. The homomorphism is called an *isomorphism* when both ϕ_v and ϕ_e are bijections.

For a set T a *multiset* of elements of T is a pair (T, f) where $f : T \to \mathbb{N}$. The number $f(t)$ is the multiplicity of $t \in T$. To ease notation, we write T for the multiset (T, f) taking the disjoint union of the copies of t in T, and hence consider it as a set when needed.

3. The combinatorial model

In this section, we introduce a model of DNA self-assembly given by flexible tiles based on graph theory.

Let H be a finite set whose elements are called *ports* and let $\theta : H \to H$ be an involution, i.e., $\theta(\theta(\mathbf{h})) = \mathbf{h}$. For each $\mathbf{h} \in H$, we call $\theta(\mathbf{h}) \in H$ the *complementary* port such that ports of codes $\mathbf{h}$ and $\theta(\mathbf{h})$ may bond. Here we encode the sticky ends of the arms of the DNA branched junction molecules with ports, elements from H. The involution θ indicates the Watson–Crick complement. We use "ports" instead of "sticky ends" to suggest that a similar approach can be taken for self-assembly processes of other chemical molecules where bonding can occur through other type of molecular bonding between the ports rather than complementarity. In that case, the involution θ can be substituted by the appropriate relation on H. In this paper, however, we concentrate on ports that simulate the sticky ends of the arms of the DNA flexible tiles. An involution θ on H is *deranging* if, for all $\mathbf{h} \in H$, $\theta(\mathbf{h}) \neq \mathbf{h}$.

Definition 1. A *port bonding system* (or PBS) is a pair (H, θ) where θ is a deranging involution θ: $H \to H$.

Given a PBS (H, θ), it follows that H has even number of elements, so we partition H in two sets H^+ and H^- such that $h \in H^+$ if and only if, $\theta(h) \in H^-$.

Definition 2. Let (H, θ) be a port bonding system. A *k-tile t* over (H, θ) is a graph with labels in H defined by $(V_t, E_t, \eta_t, \lambda_t)$ with vertices $V_t = \{ \star_t \} \cup \{ j_1, \ldots, j_k \}$, edges $E_t = \{ e_1, \ldots, e_k \}$, $\eta_t(e_i) = \{ \star_t, j_i \}$ for $i = 1, \ldots, k$, and a labeling function $\lambda_t : E_t \to H$ satisfying: $\lambda_t^{-1}(\mathbf{h}) \neq \emptyset$ implies $\lambda_t^{-1}(\theta(\mathbf{h})) = \emptyset$.

A *tile type* of a k-tile t over an PBS (H, θ) is a function $type(t) = \mathbf{t}$: $H \to \mathbb{N}$ defined with $\mathbf{t}(\mathbf{h}) = \#\{ j \in J_t \mid \lambda(j) = \mathbf{h} \}$.

A tile is a "star-like" graph, the vertex $\star_t$ of a k-tile t is called the *central* vertex of t and its degree is k. All other vertices of t have degree one and an edge incident to one of them is also incident to the central vertex. Denote the one-degree vertices of t with J_t, i.e., $J_t = \{ j_1, \ldots, j_k \}$. A k-tile can be seen as a physical representation of the k-arm branch junction molecules. A *tile* is a k-tile for some k. For each $\mathbf{h} \in H$, any tile of type $\mathbf{t}$ has exactly $\mathbf{t}(\mathbf{h})$ ports of type $\mathbf{h}$. In general, a tile type can be regarded as a multiset of port bonding types (although we will usually have $\mathbf{t}(\mathbf{h}) \leq 1$ for all $\mathbf{h} \in H$, making $\mathbf{t}$ akin to a set). Observe that $\sum_{\mathbf{h} \in H} \mathbf{t}(\mathbf{h}) = deg(\star_t)$. The definition of λ implies that either $\mathbf{t}(\mathbf{h}) > 0$ or $\mathbf{t}(\theta(\mathbf{h})) > 0$ but not both. In other words, arms of the same tile cannot bond. Note that each tile type corresponds to a unique tile, in the sense that every two tiles that are of the same tile type are isomorphic.

Definition 3. A *pot type* over a PBS (H, θ) is a set $\mathbf{P}$ of tile types over (H, θ). A *pot P* is a multiset of tiles with tile types in $\mathbf{P}$.

We presume that for each pot P there is an arbitrarily large number of tiles of each type.

Let T be a finite multiset of tiles as a disjoint union of tiles with certain tile types. Denote $\star_T = \{ \star_t \mid t \in T \}$ and with $J_T = \cup_{t \in T} J_t$. Further $E_T = \cup_{t \in T} E_t$ and η_T, λ_T are extensions of η_t and λ_t to domain E_T for each $t \in T$. Let $G_T = (\star_T \cup J_T, E_T, \eta_T, \lambda_T)$ be the disconnected graph which is the disjoint union of the tiles in T.

Definition 4. The finite multiset of tiles T *admits* a graph $\mathcal{G}_T = (T \cup J, E, \eta, \lambda)$ if $\mathcal{G}_T$ is connected and there is an onto graph homomorphism $\phi : G_T \to \mathcal{G}_T$ such that $\phi|_{\star_T}$ is a bijection onto T, $\phi|_{J_T}$ is onto J, and for each $j \in J$, $\#\phi^{-1}(j) \in \{j', j''\}$ satisfies:

$$\phi(j') = \phi(j'') \text{ implies } \lambda_t(e) = \theta(\lambda_{t'}(e')) \text{ where } \eta_t(e) = \{\star_t, j'\}, \text{ and } \eta_{t'}(e') = \{\star_{t'}, j''\}$$

Note that all vertices in J have degrees either one or two, as they are images of vertices of degree one in G_T and each can have at most two preimages. If a vertex $j \in J$ has degree two, then it is incident to two edges, one with label **h** and the other with label $\theta(\mathbf{h})$. Let $J = J' \cup J''$ where J' consists of all vertices in J of degree one and J'' consists of all vertices in J of degree two.

Lemma 3.1. *Let T be a finite multiset and $\mathcal{G}_T$ be a graph admitted by T through a homomorphism $\phi : G_T \to \mathcal{G}_T$. Then the restriction $\phi|_t$ is an isomorphism from tile t in G_T to the neighborhood N_t in $\mathcal{G}_T$.*

Proof. It follows directly from the definition of graph homomorphism and the properties of ϕ. Note that $\phi(\star_t) = t$ and for each $j \in J_t$, $\phi(j) \in J$. Each edge e in t with endpoints $\{\star_t, j\}$ maps to an edge in E with endpoints $\{t, \phi(j)\}$ as ϕ is graph homomorphism. Note that $deg(\star_t) \geq deg(t)$ since $\phi^{-1}(t) = \star_t$ and every edge incident to t is an image of an edge incident to $\star_t$. Since a tile cannot have two edges one with label **h** and the other with label $\theta(\mathbf{h})$, there are no two vertices j and j' in t that have the same image under ϕ. Hence, distinct edges in t map to distinct edges in N_t.

It follows from the definition that each tile is a graph admitted by itself. By connecting k-arm DNA branched junction molecules, two connecting arms can be considered as a new "edge" between the junctions. Similarly, such two-degree vertices in $\mathcal{G}_T$ can be ignored and the two incident edges to each of those vertices can be substituted with a single edge. This generates multigraphs which we call "complexes".

Definition 5. Let $\mathcal{G}_T$ be a graph admitted by a multiset of tiles T. The *complex $\mathcal{C}_T$ admitted by T generated by $\mathcal{G}_T$* is a multigraph obtained from $\mathcal{G}_T$ with labels in H by

(a) removal of the vertices J'', and

(b) for each $j \in J''$, removal of the pair e_1, e_2 with $\eta(e_1) = \{t_1, j\}$, $\eta(e_2) = \{t_2, j\}$, $\lambda(e_1) = \in H^+$ and adding an edge e with $\eta(e) = \{t_1, t_2\}$ with label $\lambda(e) = \lambda(e_1)$.

The *type* of the complex $\mathcal{C}_T$ is a function $type(\mathcal{C}_T) : H \to \mathbb{N}$ such that $type(\mathcal{C}_T)(\mathbf{h}) = \#\{\, e \in E \mid \exists j \in J',\ \eta(e) = \{t, j\}, \lambda(e) = \mathbf{h}\,\}$.

Again, for each tile t $J''_t = \emptyset$, so a tile can be considered as a complex admitted by itself, in which case the type of the tile is the same as the type of the complex. We say that $\mathcal{C}_T$ is admitted by T if there is a graph $\mathcal{G}_T$ admitted by T that generates $\mathcal{C}_T$. The type of a complex $\mathcal{C}_T$ indicates the number for each of the types of its free ports.

Definition 6. The *set of complexes admitted by a pot type* **P** is the set

$$\{\, \mathcal{C}_T \mid \exists T\ \forall t \in T, type(t) \in \mathbf{P},\ \text{and } \mathcal{C}_T \text{ is admitted by } T\,\}.$$

We assume that the assembly process occurs in an extremely dilute solution, so that when two complexes meet, *all* of their complementary free sticky ends join up before any other complexes are encountered, so we can treat the complex as if at no time does it have complementary free ports. (This is where the flexibility of the tiles is so critical.) Thus:

Definition 7. A complex $\mathcal{C}$ admitted by tiles T is *stable* if, for each $\mathbf{h} \in H$, either $type(\mathcal{C})(\mathbf{h}) = 0$ or $type(\mathcal{C})(\theta(\mathbf{h})) = 0$ or both.

Note: In this paper, all complexes are stable unless otherwise indicated.

Unlike the case of tile types when each tile type determines unique (up to isomorphism) tile, non-isomorphic complexes can have the same type.

Consider the example shown in Fig. 2. Three tiles shown in (a) are (maximally) connected to produce two non-isomorphic complexes of the same type admitted by the three tiles. The ports are shown with different colors and shapes, labels on the edges of the tiles indicate the port types. The graph homomorphism from the three tiles to the two graphs admitted by the tiles shown in (b) is such that $j_1, j_2 \mapsto j$, $k_1, k_2 \mapsto k$, $p_1, p_2 \mapsto p$ and $n_2, n_3 \mapsto n$ for the graph to the left and $j_1, j_2 \mapsto j$, $k_1, k_2 \mapsto k$, $p_2, p_3 \mapsto p$ and $n_2, n_3 \mapsto n$ for the graph to the right. The two corresponding complexes admitted by the tiles are depicted in (c). They are obtained from the graphs in b by removing the two-degree vertices j, k, p and n and connecting the corresponding two incident edges into a single edge.

Definition 8. A complex is called *complete* if it has no free ports, i.e., if for all port bonding types **h**, $type(\mathcal{C})(\mathbf{h}) = 0$. The set of all complete complexes admitted by a pot type **P** is called the *output of the pot type* **P** and is denoted with $\mathcal{O}(P)$.

The output of the pot type **P** is the set of complexes that have reached its "final stage" of the assembly, in the sense that they cannot change their structure any more.

Designing pot types that encode computational problems such that the solution of the problem can be realized within the output of the pot is the topic of the following sections.

4. Examples

We turn to the primary issue of this article: representing a query as a pot type, where the question "does a solution exist?" can be reduced to "can a suitable complete complex be assembled?"

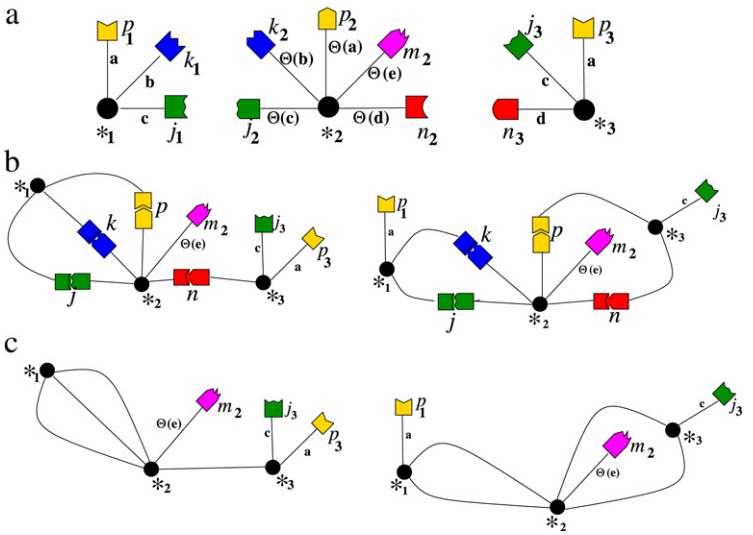

Fig. 2. (a) Three tiles, with their central vertices shown as black circles, with their free arm vertices schematically presented by different colors and shapes. The complementary ports have compatible shapes, but the same colors. Labels on the edges indicate the port types. (b) Two different graphs admitted by joining three tiles in (a) by connecting complementary ports. (c) Two (incomplete) complexes corresponding to the graph structures in (b) admitted by tiles in (a). The complexes (c) are non-isomorphic but of the same complex type.

4.1. Cycles in graphs

A *cycle* in a (not necessarily labeled) multigraph $G = \langle V, E, \eta\rangle$ is a sequence of distinct edges and vertices $v_0, e_1, v_1, \ldots, e_k, v_k = v_0$ such that $\eta(e_i) = \{v_{i-1}, v_i\}$ for $i = 1, \ldots, k$. Consider the boolean query "the graph contains a cycle" (or "the graph is not acyclic"). We present an algorithm for taking an input G and converting it to a pot type $\mathbf{P}(G)$ such that G has a cycle if and only if $\mathbf{P}(G)$ can generate a complete complex. Such a complete complex can in fact exhibit a cycle witnessing the truth of the query.

Start with a PBS:

$$H = \{(v, e) \in V \times E \mid v \in \eta(e)\}, \text{ and}$$
$$\theta : H \to H \qquad \theta((v, e)) = (v', e) \quad \text{if } \{v, v'\} = \eta(e).$$

From this set of port bonding types, for each vertex $v \in V$ of degree ≥ 2 and distinct edges $e_1, e_2 \in E$ incident to v, create a 2-tile of type $\mathbf{t} = \mathbf{t}_{v,e_1,e_2}$ such that $\mathbf{t}(v', e) = 1$ if $v' = v$ and $e \in \{e_1, e_2\}$, and $\mathbf{t}(v', e) = 0$ otherwise. Note that for any tile type $\mathbf{t}$, if $\mathbf{t}((v_1, e_1)), \mathbf{t}((v_2, e_2)) > 0$, then $v_1 = v_2$, and $\sum_{\mathbf{h}\in H} \mathbf{t}(\mathbf{h}) = 2 = deg(\star_t)$ for every tile t of type $\mathbf{t}$. A vertex is not represented by any tiles if it is of degree less than 2. The complete complexes admitted by such a pot are cycles of tiles, t_0 connected to t_1 connected to t_2 connected to ... connected to t_k connected back to t_0. Each tile t_i in the complete complex represents a vertex v_i, and each connection from a tile t_i to a tile t_{i+1} will be represented by an edge from v_i to v_{i+1} (and similarly for the connection from v_k to v_0).

While it is straightforward to verify that every cycle has a corresponding complete complex, the converse is not true. A complete complex could have several tiles representing the same vertex, and thus actually representing a "closed walk" on the graph, i.e., a sequence $v_0, v_1, v_2, \ldots, v_n, v_0$, where each pair of successive vertices are adjacent in the graph, and (noting that a tile does not have two ports for the same edge, and hence the (acyclic) complex cannot connect two tiles representing the same vertex) each pair of successive edges are adjacent on the graph. On the other hand, if we choose the least k such that $v_0 = v_k$ in the closed walk, $v_0, v_1, \ldots, v_k, v_0$ *is* a cycle in the graph. Thus we can conclude:

- For a graph G the generated pot of type $\mathbf{P}(G)$ admits complete complexes if and only if the graph G admits cycles.
- If we conducted this experiment, and we got complete complexes, these complete complexes would represent closed walks, which in turn would contain cycles. The minimal complete complexes actually would represent cycles.

This is an example of a problem in which the question "does the inputted structure satisfy the given property?" can be answered simply by checking if complete complexes are admitted by the pot, i.e., just by checking whether $\mathcal{O}(P) = \varnothing$.

This criterion may not apply to other problems.

4.2. The 3SAT problem

The *satisfiability query* (SAT) has as its input a boolean formula φ, and SAT is TRUE if there is an assignment of {TRUE, FALSE} (or $\{T, F\}$) to the propositional variables in φ that would assign φ the value TRUE. (This is the archetypic NP-complete

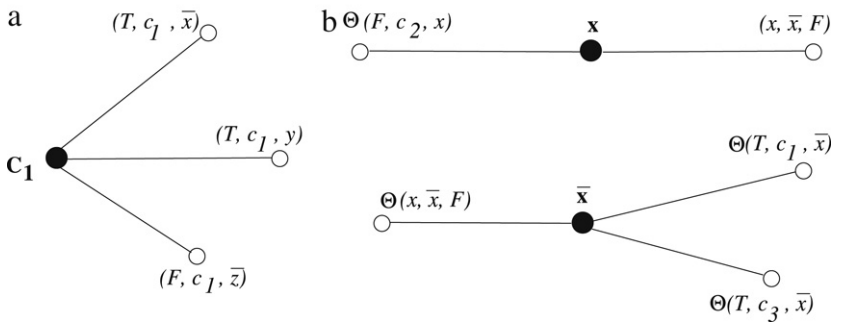

Fig. 3. (a) A tile for clause c_1 for φ in Formula (1) with assignment (F, T, T). (b) Tiles corresponding to variable x and $\bar{x}$ (with value F for x) for φ in Formula (1).

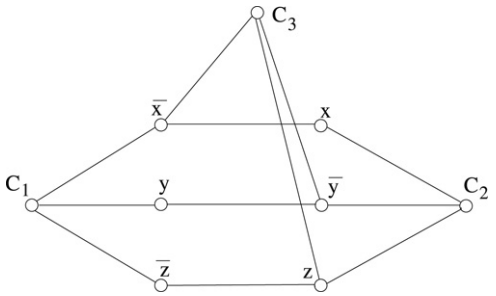

Fig. 4. A complete complex for φ in Formula (1). Labels of the edges indicating the truth assignment of the variables are omitted.

problem [7].) Writing "+" for "or" and concatenation for "and", a formula φ is in *conjunctive normal form* (CNF) if it can be expressed as $\varphi = c_1 c_2 \cdots c_r$ where each c_i is a clause of the form $(a_1 + \cdots + a_k)$, and each a_i is a *literal*, i.e., either a variable or a negation of a variable (we represent negation by overbars). It is in *k-conjunctive normal form* (k-CNF) if each conjunctive clause contains at most k literals. The satisfiability problem for 3-CNF formulas is known as 3SAT, and is also known to be NP-complete [7].

Consider the example:

$$\varphi = (\bar{x} + y + \bar{z})(x + \bar{y} + z)(\bar{x} + \bar{y} + z). \tag{1}$$

This 3-CNF φ in Formula (1) has value T for the assignments $(x, y, z) \in \{(F, F, F), (T, T, T), (T, F, F), (F, T, T), (F, F, T)\}$, and F for any other assignment. Hence it is satisfiable.

For each formula φ we associate a pot type $\mathbf{P}(\varphi)$ in the following way. The set of port bonding types is

$$H = \{(\iota, c, a), \theta(\iota, c, a) \mid \iota \in \{T, F\}, c \text{ a clause}, a \text{ a literal of } c\} \cup \{(x, \bar{x}, \iota), \theta(x, \bar{x}, \iota) \mid x \text{ is a variable}, \iota \in \{T, F\}\}.$$

The tiles differ in the assignment of the free ports. For each clause c there are seven 3-tile types with ports of types (ι, c, a). Each of the seven tile types corresponds to one the seven truth assignments of the literals in c that make the clause "true". Thus, the tile type corresponding to clause $c_1 = (\bar{x} + y + \bar{z})$ for φ in Formula (1) with assignment $x \to F, y \to T, z \to T$ has ports $(T, c_1, \bar{x})$, (T, c_1, y), and $(F, c_1, \bar{z})$ for $(\bar{x} \to T, y \to T, \bar{z} \to F)$, and is depicted in Fig. 3(a).

For each variable x, if x appears in s clauses, we associate x with two tile types each with $s + 1$ arms. Each tile corresponds to one of the two possible truth values of the variable, T or F. Each of the ports on s of the arms is complementary to the corresponding port in one of the corresponding clause tiles (encoding θ(truth value, clause, variable)); the additional $s + 1$st port is connected to the tile of the negation of the variable and serves to ensure opposite truth value to the complement of the variable. Similarly, two tiles are encoded for the complement of each of the variables. The tiles corresponding to x and $\bar{x}$ for φ in Formula (1) assigning value F to x and T to $\bar{x}$ are depicted in Fig. 3 (b).

For φ in Formula (1), an example of a complete complex admitted by the tiles (suppressing edge labels and the truth assignments to the variables) is presented in Fig. 4. All edges are obtained by gluing complementary ports. It is not difficult to see that any complete complex has to have at least one tile for each clause, and at least one tile for each literal (a variable or a negation of a variable). A complete complex with exactly one copy of the tiles for each of the clauses, variables and negation of the variables exists if and only if φ is satisfiable. The labels of each such complex encode the truth assignment of the variables.

On the other hand, if there is no satisfying truth assignment of the variables, there may still be large complete complexes. For simplicity, consider 2-CNF formulas. Then, for example, $\alpha = (x + y)(x + \bar{y})(\bar{x} + y)(\bar{x} + \bar{y})$ is not satisfiable. Label clauses $c_1 = (x + y)$, $c_2 = (x + \bar{y})$, $c_3 = (\bar{x} + y)$ and $c_4 = (\bar{x} + \bar{y})$. Tiles equivalent to the ones described for φ in Formula (1) can

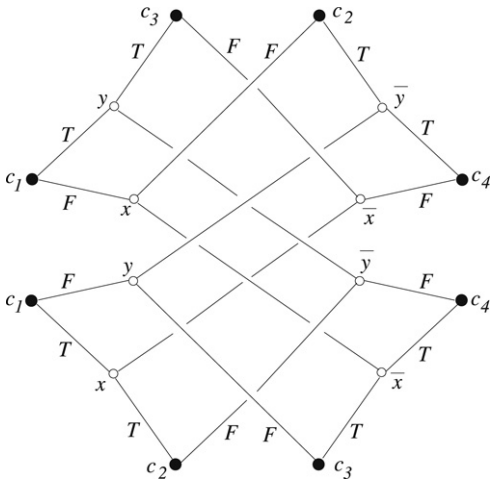

Fig. 5. A complete complex with minimal number of tiles for $\mathbf{P}(\alpha)$. Its size is twice the size of the complex which would have corresponded to the solution of the satisfiability problem.

admit a complete complex as shown in Fig. 5. Each clause in this case is represented by a 2-tile instead of a 3-tile, as each clause contains only two literals. The labels of the edges indicate an example of "truth assignments" encoded through the ports for the literals labeling the incident vertices. The complete complex in Fig. 5 contains two tiles for each of the clauses, variables and negation of the variables. Although this is a complex with minimal number of tiles, and therefore the formula is not satisfiable.

In this example, no matter what we input, we can wind up with complete complexes admitted by the pot. The only way to tell if the inputted 3-CNF formula is satisfiable is to determine if there are sufficiently small complete complexes in the pot. Directly from the construction we have:

Lemma 4.1. *If a CNF formula φ has k clauses and n variables, $\mathcal{O}(P(\varphi))$ contains a complex with $k + n$ tiles if and only if, φ is satisfiable.*

A self-assembly computation is obtained by putting a mixture containing tiles of certain types into a pot and identifying the complete complexes that are formed. At the end of the experiment, there may be some complete complexes of the desired form, some other complete complexes, and some incomplete complexes. The above lemma induces the concept for satisfiable pots within a given bound.

Definition 9. A pot type is *weakly satisfiable within bound b* if it admits a complete complex of at most b tiles.

A pot type is weakly satisfiable[1] if it admits a complete complex.

Therefore, a CNF formula φ with k clauses and n variables is satisfiable if and only if, $\mathcal{O}(P(\varphi))$ is weakly satisfiable within bound $n + k$. It is well known that SAT is NP-complete, meaning that every other NP problem can be reduced within a polynomial time to a SAT problem. Therefore, every NP problem is weakly satisfiable within a polynomial bound. This can be generalized to a larger class of problems as we show in Section 6.

5. Identifying contents of the pot

In this section we formalize the situation identified in the example of SAT, meaning, we define pots such that every complete complex admitted by the pot maps homomorphically onto a predetermined graph.

Definition 10. Let $G = (V, E, \eta, \lambda_E)$ be a connected multigraph with labeling λ_E being the identity on the edges. Let (H, θ) be a port bonding system, and $\mu : H^* \to E$ be an onto function for some $H^* \subseteq H$. We say that the pair (G, μ) is *an indexing system for a pot type P* over (H, θ) if, for every $\mathcal{C} \in \mathcal{O}(\mathbf{P})$ with labeling function λ, there is an onto homomorphism ϕ from the subgraph of $\mathcal{C}$ generated by the edges with labels in H^* onto G satisfying $\lambda_E(\phi(e)) = \mu(\lambda(e))$.

Notice that in the case of SAT, for a CNF formula φ, an index system is a pair (G, μ) where $G = (C \cup X, E, \eta, \lambda_E)$ with $C = \{ c \mid c \text{ is a clause in } \varphi \}$, $X = \{ x \mid x \text{ is a literal in } \varphi \}$ and $E = \{ (c, a) \mid a \text{ is a literal in } c \} \cup \{(x, \bar{x}) \mid x \text{ is a variable in } \varphi \}$, the endpoints are $\eta(c, a) = \{c, a\}$ and $\eta(x, \bar{x}) = \{x, \bar{x}\}$; the function μ is defined on $H^* = H^+$ such that $\mu : H^+ \to E$ is onto with $(\iota, c, a) \overset{\mu}{\mapsto} (c, a)$ and $(x, \bar{x}, \iota) \overset{\mu}{\mapsto} (x, \bar{x})$. For formula φ in Formula (1) the graph G of its index system is depicted

[1] We can classify a pot as *weakly satisfiable* if it admits a nonempty complete complex, *satisfiable* if it admits a complete complex with all port codes used, and *strongly satisfiable* if it admits a complete complex with all tile types used. These are distinct notions: see [20].

in Fig. 4. To see that the pair (G, μ) defined above is an indexing system for φ, observe that every complete complex of $\mathbf{P}(\varphi)$ has to contain a tile for every clause and every variable. Therefore we map clause tiles from complete complexes to the corresponding clause vertex in G. The edges are mapped accordingly (preserving the homomorphism properties) and the labeling map follows μ.

Observe that if $H^* \subsetneq H$, the complex may not have edges that are labeled with codes not in H^* and thus not appearing in the index system. An index system merely provides a way for traversing the tiles of a complex.

On the other hand, in the case of checking for a cycle in a graph, there are instances of graphs for which the pot type constructed for the solution of the problem has no indexing system.

Lemma 5.1. *Let* $\mathbf{P}$ *be a pot type over port bonding system* (H, θ) *and let* (G, μ)*, where* $G = (V, E, \eta, \lambda_E)$ *and* $\mu : H^* \to E$*, be an indexing system for* $\mathbf{P}$*. If each tile with type in* $\mathbf{P}$ *has at least one port of a type in* H^* *and* μ *is injective, then any complete complex admitted by* $\mathbf{P}$ *has* $\#V \cdot n$ *tiles, for some integer* n*.*

Proof. It suffices to prove that for any complete complex $\mathcal{C}$, and an onto homomorphism $\phi : \mathcal{C}^* \to G$ ($\mathcal{C}^*$ being a subcomplex of $\mathcal{C}$ generated by edges labeled in H^*) for every $v, w \in V$, $\#\phi^{-1}(v) = \#\phi^{-1}(w)$. We prove this by induction on the distance from v to w. Note that the set of tiles (vertices) in $\mathcal{C}^*$ and $\mathcal{C}$ are the same, except $\mathcal{C}^*$ may have a lower number of edges. This follows from the assumption that each tile contains a port in H^*.

If the distance from v to w is 0, i.e., $\mathrm{dist}(v, w) = 0$, then $v = w$. For the inductive step, suppose that for any v, u of distance d apart, if there are n tiles in the preimage of v then there are n tiles in the preimage of u. Assume now that $\mathrm{dist}(v, w) = d + 1$, and there are n tiles in the preimage of v. Then w is adjacent to some vertex u of distance d from v, $\mathrm{dist}(u, v) = d$. By the inductive hypothesis, there are n tiles in the preimage of u. As u is adjacent to w in G, every tile in the preimage of u connects to a tile in the preimage of w and vice versa. Note that the edge e connecting u and w in G has a label (the edge itself) which uniquely determines this edge. Thus each one of the vertices in the preimage of u has an edge that maps into the edge e with labels in H^*. And each one of the vertices (tiles) in the preimage of w is incident to an edge that maps onto e. As every $h \in H^*$ can be a port of at most one arm in a tile, $\phi^{-1}(e)$ induces a one-to-one correspondence between $\phi^{-1}(u)$ and $\phi^{-1}(w)$.

Therefore, there are precisely n preimages of w.

6. Computing by flexible tiles

In this section we define the complexity classes for flexible tile computable problems and we show that these can be associated with the standard complexity classes of languages. For the purpose of fixing notation and completeness of the paper we recall the basic definitions of Turing machines and their complexity classes.

Fix a finite alphabet Σ; and let $\square \notin \Sigma$ be the symbol used for "blank". We use the standard hierarchy of complexity classes of languages, i.e., subsets of Σ^*.

Definition 11. A *Deterministic Turing Machine (DTM)* is a tuple $\mathcal{M} = (Q, \Sigma, \delta, q_0, q_\infty, \square)$, where:

- The set Q is the set of states; q_0 is the *initial state* and q_∞ is the *terminal (accept) state*.
- The *transition function* δ is a partial function $(Q \times (\Sigma \cup \{\square\})) \to (Q \times \Sigma \times \{L, R\})$, where for any state $q \neq q_\infty$ and character $\sigma \in \Sigma$, $\delta(q, \sigma) = (\delta_Q(q, \sigma), \delta_\Sigma(q, \sigma), \delta_m(q, \sigma))$. If the tapehead is scanning $\sigma \in \Sigma \cup \{\square\}$ while the machine is in state q, it writes $\delta_\Sigma(q, \sigma)$ on its current square, moves $\delta_m(q, \sigma) =$ *L*eft or *R*ight, and goes into state $\delta_Q(q, \sigma)$; $\delta(q_\infty, \sigma)$ is undefined for all $\sigma \in \Sigma$.
  When $\delta_Q(q, \sigma) = q'$, $\delta_\Sigma(q, \sigma) = \sigma'$ and $\delta_m(q, \sigma) = \xi$, we say that $(q, \sigma) \to (q', \sigma', \xi)$ is a *transition step*.
- The machine starts with the tapehead at the leftmost square of the input string while in state q_0; if it enters state q_∞, then the computation halts and the input is accepted: the machine *accepts* if and only if it halts.

The machine is a *Non-Deterministic Turing Machine (NTM)* if, for each state q and character σ, $\delta(q, \sigma)$ is a nonempty subset of $Q \times \Sigma \times \{L, R\}$ such that a transition is a step $(q, \sigma) \to (q', \sigma', \xi)$ if $(q', \sigma', \xi) \in \delta(q, \sigma)$. The machine halts only in state q_∞ and $\delta(q, \sigma) = \emptyset$ if and only if, $q = q_\infty$.

A *configuration* of a Turing machine $\mathcal{M}$ is a tuple $(q, m, l, \mathbf{s})$, where q is the state of $\mathcal{M}$, m is the position of the tapehead, l is the tape position of the left-most non-blank character, and $\mathbf{s}$ is the non-blank string currently on the tape. A configuration is *initial* if $q = q_0, m = 1, l = 1$, and $\mathbf{s}$ is the input string (the rest of the tape being blank at the beginning of the computation). Thus, a *computation* is a sequence $C_0, C_1, \ldots$, where C_0 is an initial configuration, and for each t, C_{t+1} follows from C_t after a transition step. The computation terminates only if it halted: if C_t is halted, then $C_{t+1} = C_t$. We say that the least such t is the *computation time* of $\mathcal{M}$ for $\mathbf{s}$ denoted $\mathrm{TIME}_{\mathcal{M}}(\mathbf{s})$. Note that if C_t is not halted, then as the unhalted tapehead always moves, $C_{t+1} \neq C_t$. If $\mathcal{M}$ admits no halting computation on input $\mathbf{s}$, we say that $\mathcal{M}$ rejects $\mathbf{s}$ and write $\mathrm{TIME}_{\mathcal{M}}(\mathbf{s}) = \infty$.

Let $\mathrm{TIME}_{\mathcal{M}}(n) = \max\{\mathrm{TIME}_{\mathcal{M}}(\mathbf{s}) < \infty : |\mathbf{s}| = n\}$.

We employ two kinds of computations: those that (deterministically) compute functions from strings to strings, and those that accept (or fail to accept) strings.

Let $\mathcal{M}$ be a Turing machine and $L_{\mathcal{M}}$ be the set of strings (the language) *accepted* by $\mathcal{M}$. (Note that if $\mathcal{M}$ is deterministic then there is only one computation of $\mathcal{M}$ on a given input $\mathbf{s}$).

If $\mathcal{M}$ is deterministic and halts on every input, let $F_{\mathcal{M}} : \Sigma^* \to \Sigma^*$ be the function such that given input $\mathbf{s}$, $\mathcal{M}$ outputs $F_{\mathcal{M}}(\mathbf{s})$. We say that $F_{\mathcal{M}}$ is the function computed by $\mathcal{M}$.

For any function $f\colon \mathbb{N} \to \mathbb{N}$, let $\mathrm{DTIME}(f) = \{\mathcal{M} \mid \mathcal{M}$ a deterministic Turing machine and $\forall n, \mathrm{TIME}_{\mathcal{M}}(n) \leq f(n)\}$. Given a class $\mathcal{F}$ of functions $\mathbb{N} \to \mathbb{N}$, let $\mathrm{DTIME}(\mathcal{F}) = \bigcup_{f\in\mathcal{F}} \mathrm{DTIME}(f)$. A language is $\mathrm{DTIME}(\mathcal{F})$-*computable* if it is accepted by a deterministic Turing machine in $\mathrm{DTIME}(\mathcal{F})$, while a function is $\mathrm{DTIME}(\mathcal{F})$-*computable* if it is computable by a deterministic Turing machine in $\mathrm{DTIME}(\mathcal{F})$.

A set $\mathcal{F}$ of functions $f\colon \mathbb{N} \to \mathbb{N}$ is *polynomial closed* if for each $f \in \mathcal{F}$ and each polynomial $p\colon \mathbb{N} \to \mathbb{N}$, $p \circ f \in \mathcal{F}$. In particular, the "polynomial-time computable functions" are those that are computed by deterministic Turing machines in $\mathrm{DTIME}(\mathcal{F})$, where $\mathcal{F}$ is the set of polynomials on $\mathbb{N} \to \mathbb{N}$. This class is denoted PTIME or just P.

By substituting deterministic with non-deterministic Turing machines, $\mathrm{NTIME}_{\mathcal{M}}(n)$, $\mathrm{NTIME}(f)$ and $\mathrm{NTIME}(\mathcal{F})$ are defined. In particular, the "non-deterministic polynomial-time computable queries" are those corresponding to domains of nondeterministic Turing machines in $\mathrm{NTIME}(\mathcal{F})$, where $\mathcal{F}$ is the set of polynomials on $\mathbb{N} \to \mathbb{N}$; this class is often denoted NPTIME or just NP.

Remark. One of the characteristics of NPTIME is that it seems to consist of those queries expressible in the form "there exists a witness structure that articulates with the input in such-and-such a way". This is seen in the classical results of [8] and [33] (and in the representation of NPTIME enumerated in [12]), in which NPTIME is shown to be expressible in the logical form "$\exists S\theta(S)$" which is true of an input structure $\mathfrak{I}$ if $\mathfrak{I}$ admits some structure or structure-like object $S^{\mathfrak{I}}$ such that $\theta(S^{\mathfrak{I}})$ is true of $\mathfrak{I}$. The main result of this paper could be regarded as another representation theorem of this sort.

6.1. Pot types representing NTIME

In this subsection, we describe a $\mathrm{DTIME}(\mathcal{F})$ computation that takes a nondeterministic Turing machine $\mathcal{M}$ with an input $\mathbf{s}$ and produces a pot type $\mathbf{P}$ such that $\mathcal{M}$ accepts $\mathbf{s}$ if and only if $\mathbf{P}$ admits a complete complex with predetermined size. In particular, if $\mathcal{M}$ has an accepting computation of $\mathbf{s}$ of length at most $f(|\mathbf{s}|)$, then the acceptable complete complexes have at most about $2f(|\mathbf{s}|)^2$ tiles.

As is usually done, the complete complex representing the Turing machine computation simulates the configuration steps, starting with C_0 and ending with C_t for t being the computation time. Each tile in the complete complex represents one of the locations of the tape at a given time.

More precisely, each tile represents one tape square at a position m at a time t. At that time and position, there is a symbol σ in the square, and we define $\xi_0 = 1$ if the tapehead is at this square at time t, and $\xi_0 = 0$ otherwise. If $\xi_0 = 1$, we define ξ_- and ξ_+ to represent the previous and next moves of the tapehead. The head had moved from the left ($\xi_- = L$) or the right ($\xi_- = R$), and the head will move either to the left ($\xi_+ = L$) or the right ($\xi_+ = R$). And if the machine entered square m in a state q at time t, it will transition to a state q' at time $t + 1$. Finally, if the computation terminated at time t, $q' = q_\infty$ and $\xi_+ = 0$.

Definition 12. A *tile index set for space bounds* (l, r) *for* $l < 0$ *and* $r > 0$ *and time bound* $t_{\max}$ is a set of tuples

$$(\sigma, m, t, (\xi_-, \xi_0, \xi_+), (q, \sigma', q')),$$

such that $\sigma \in \Sigma \cup \{\square\}$; $l \leq m \leq r$; $0 \leq t \leq t_{\max}$; $\xi_0 \in \{0, 1\}$ and $\xi_-, \xi_+ \in \{L, R, 0\}$; and $q, q' \in Q$ satisfying:

- If $\xi_0 = 0$, then $\xi_- = \xi_+ = 0$.
- If $\xi_- \neq 0$ and $q = q_\infty$, then $q' = q_\infty$ and $\xi_0 = \xi_+ = 0$.

These indices are supposed to represent legal positions of a Turing machine.

Definition 13. Given a tile index set $(\sigma, m, t, (\xi_-, \xi_0, \xi_+), (q, \sigma', q'))$ and a (nondeterministic) Turing Machine $\mathcal{M}$, the index is consistent with $\mathcal{M}$ (i.e., with its transition function $\delta = (\delta_Q, \delta_\Sigma, \delta_m)$) if, whenever $\xi_0 = 1$ and $q \neq q_\infty$, then $(q, \sigma) \to (q', \sigma', \xi_+)$. We take ξ_- to be the *previous move*.

6.1.1. Port bonding system

There are eight kinds of ports, of which any particular tile may have from two to six, in positions which we label as a compass: N, NE, E, SE, S, SW, W, and NW. Recall that a tile represents a particular mth square of the Turing machine at a particular time t. The idea is to force the formation of successive rows of tiles representing successive configurations of the Turing Machine, starting with the initial (northernmost) row, and heading south. Consider the tile for the mth square at time t.

- The N and S directions are for ports on the arms of the (m, t)th tile connecting it to the $(m, t-1)$th (same location of the tape but one move before) and $(m, t+1)$st (same location but one move into the future).
- The E and W directions are for ports of the (m, t)th tile connecting to the $(m-1, t)$th and $(m+1, t)$th tiles representing the neighboring squares at the same time t (i.e., same configuration).

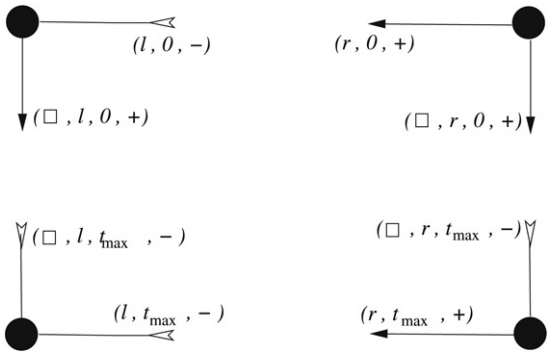

Fig. 6. Corner tiles.

The edge bonds N–S and E–W form a grid that fixes the overall graphical structure of the resulting complex: this grid is the index graph of the structure, of port types N and E, as enumerated below. But there are additional diagonal edges to track the motion of the tapehead.

- The NW port of (m, t)th tile, if any, extends to the $(m-1, t-1)$th tile if the head was on the $m-1$st tile at time $t-1$ and then moved right. Similarly, the NE port, if any, extends to the $(m+1, t-1)$th tile if the head was on the $m+1$st tile at time $t-1$ and then moved left.
- And the SW port of (m, t)th tile, if any, extends to the $(m-1, t+1)$th tile if the head was on the mth tile at time t and then moved left; similarly, the SE port, if any, extends to the $(m+1, t+1)$th tile if the head was on the mth tile at time t and moved right.

If the computation is halted, the identity of the square with the tapehead is no longer relevant, and so disappears from port codes.

Therefore, the set of port types H is defined with: $H = N \cup S \cup E \cup W \cup NE \cup NW \cup SE \cup SW$ where:

$$
\begin{aligned}
N &= \{(\sigma, m, t, -) \mid \sigma \in \Sigma, l \le m \le r, 0 < t \le t_{\max}\}\\
S &= \{(\sigma, m, t, +) \mid \sigma \in \Sigma, l \le m \le r, 0 \le t < t_{\max}\}\\
W &= \{(m, t, +) \mid l < m \le r, 0 \le t \le t_{\max}\}\\
E &= \{(m, t, -) \mid l \le m < r, 0 \le t \le t_{\max}\}\\
NW &= \{(q, m, t, R, -) \mid q \in Q,\ l < m \le r,\ 0 < t \le t_{\max}\}\\
NE &= \{(q, m, t, L, -) \mid q \in Q,\ l \le m < r,\ 0 < t \le t_{\max}\}\\
SW &= \{(q, m, t, L, +) \mid q \in Q,\ l < m \le r,\ 0 \le t \le t_{\max}\}\\
SE &= \{(q, m, t, R, +) \mid q \in Q,\ l \le m < r,\ 0 \le t \le t_{\max}\}
\end{aligned}
$$

And the involution $\theta : H \to H$ satisfies:

$$
\begin{aligned}
\theta((\sigma, m, t, -)) &= (\sigma, m, t-1, +) & \theta((m, t, -)) &= (m+1, t, +)\\
\theta((q, m, t, R, -)) &= (q, m-1, t-1, R, +) & \theta((q, m, t, L, +)) &= (q, m-1, t-1, L, -)
\end{aligned}
$$

And we will fix the grid using the indexing system with $H^* = S \cup W$.

6.1.2. Tiles

Given the tile index set for space bounds (l, r) for $l < 0$ and $r > 0$ and time bound $t_{\max}$ consistent with the Turing machine $\mathcal{M}$ we define tiles in the following way. The tiles have degrees from 2 to 6. For index $\alpha = (\sigma, m, t, (\xi_-, \xi_0, \xi_+), (q, \sigma, q'))$ the tile t_α has a central vertex $\star_\alpha$ with degrees:

$$
deg(\star_\alpha) = \begin{cases}
2 & \text{if } m \in \{l, r\} \text{ and } t \in \{0, t_{\max}\} \quad \text{[if a corner tile;]}\\
3 & \text{if } m \in \{l, r\} \text{ xor } t \in \{0, t_{\max}\} \quad \text{[if a side tile;]}\\
4 & \text{if } \xi_0 = 0 \quad \text{[if an interior tile with no tapehead;]}\\
5 & \text{if } \xi_1 = 1 \text{ and } t = 0 \quad \text{[if computation just starting, at initial square;]}\\
6 & \text{if } q' = q_\infty \quad \text{[if the computation halts;]}\\
7 & \text{if } \xi_0 = 1 \text{ and } q' \neq q_\infty \quad \text{[if tapehead present, not halted.]}
\end{cases}
$$

The corner tiles with two arms together with their ports are depicted in Fig. 6.

The side tiles (top, bottom, left and right) which represent portions of the configurations of the tape away from the tapehead are tiles with three arms and they are depicted in Fig. 7. The central portion of Fig. 7 contains two copies of tiles that are at position m at $t-1$st and tth configuration away from the position of the movement of the tape. These tiles are used to "remember" the symbol at position m on the tape from one time-step to another.

Suppose that the head is at location m at $t-1$st time configuration so that $\xi_0 = 1$, so that the head, just having entered state q and arrived at square m, reads in σ, applies some rule $(q, \sigma) \to (q', \sigma', \xi)$ and enters state q', writes σ', and either

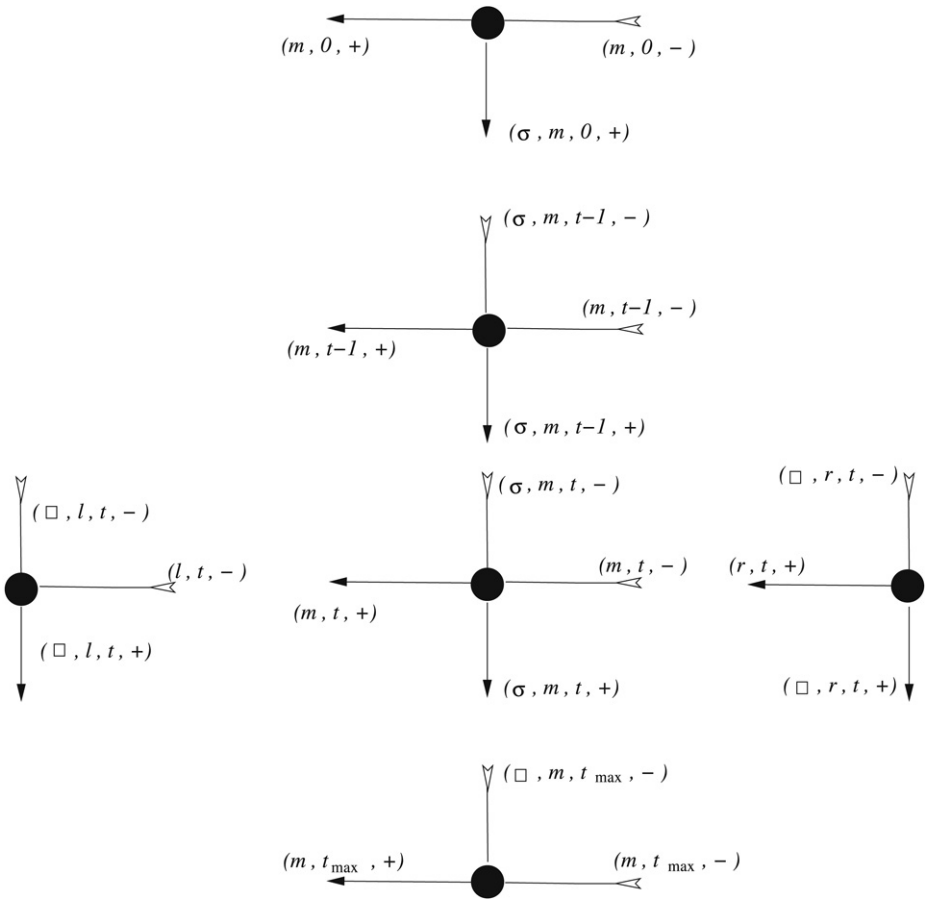

Fig. 7. Side tiles and central tiles that represent part of the configuration that is away from the movement of the tapehead.

moves left ($\xi = L$), right ($\xi = R$), or, if $q' = q_\infty$, halts. Without loss of generality, assume that the head had arrived from the left and moves to the left. Then this situation is represented with tile $(\sigma, m, t-1, (L, 1, L), (q, \sigma, q'))$ at position m at time $t-1$. This tile has six arms: N, S, W, E, NE, and SW. The ports of N, E, and W are similar, except that $\xi_0 = 0$ and tapehead information is omitted as not applicable, e.g., the S port is now $(\sigma', m, t-1, +)$ instead of $(\sigma, m, t-1, +)$. This is depicted in Fig. 8 with the top right tile.

In general, a tile represents a tapehead action if it has an arm with either a NW or a NE port, $(q, m, t-1, \xi_-, -)$, (to connect to a port $(q, m+1, t, \xi_-, +)$), NW if $\xi_- = R$ and NE if $\xi_- = L$ (treating $m + L$ as $m - 1$ and $m + R$ as $m + 1$). And there is an arm with either a SW or a SE port, $(q', m, t-1, \xi_+, +)$ (to connect to a port $(q', m+1, t, \xi_+, -)$), SW if $\xi_+ = L$ and SE if $\xi_+ = R$. *But*: the tile has no SW or SE port if, in state q and reading in σ, the computation *halts*.

6.1.3. Pot type

Definition 14. Given $l < 0, r > 0, t_{\max}$, a Turing machine $\mathcal{M}$, and an input string $\mathbf{x} = x_0 x_1 \cdots x_{n-1} \in \Sigma^n$, where $n \leq r + 1$, the *simulation pot (type)* $\mathbf{P} = (\mathcal{M}, \mathbf{x}, l, r, t_{\max})$ is the set of all tiles from an index set tiles for space bounds (l, r) and time bound $t_{\max}$ consistent with $\mathcal{M}$ such that for $t = 0$, the tiles t_α are included in $\mathbf{P}$ if and only if:

- $(\square, m, 0, (0, 0, 0), (0, 0, 0))$ for $m < 0$ or $m \geq n$, or
- $(x_0, 0, 0, (0, 1, \xi_+), (q_0, \sigma', q'))$ for $m = 0$ and $(q_0, x_0) \to (q', \sigma', \xi_+))$, or
- $(x_m, m, 0, (0, 0, 0), (0, 0, 0))$ for $1 \leq m \leq n - 1$.

Observe that the number of tile types in $\mathbf{P}$ is polynomial with respect to $\#Q, \#\Sigma, |\mathbf{x}|, l, r, t_{\max}$.

Lemma 6.1. *Let* $\mathbf{P} = (\mathcal{M}, \mathbf{x}, l, r, t_{\max})$ *be defined as in Definition 14. Consider the graph* $G_{grid} = (\hat{V}, \hat{E}, \hat{\eta}, \lambda_E)$ *defined with* $\hat{V} = \{(m, t) \mid l \leq m \leq r, 0 \leq t \leq t_{\max}\}$, *edges* $\hat{E} = E_{NS} \cup E_{WE}$ *where* $e^{\downarrow}_{(m,t)} \in E_{NS}$ *iff,* $\eta(e^{\downarrow}_{(m,t)}) = \{(m, t), (m, t+1)\}$ *for* $l \leq m \leq r$ *and* $0 \leq t < t_{\max}$ *and* $e^{-}_{(m,t)} \in E_{WE}$ *iff,* $\eta(e^{-}_{(m,t)}) = \{(m, t), (m+1, t)\}$ *where* $l \leq m < r$ *and* $0 \leq t \leq t_{\max}$. *Take* $H^* = S \cup W \cup E$ *and* $\mu : H^* \to \hat{E}$ *such that S maps onto edges* E_{NS}, *i.e.,* $\mu((\sigma, m, t, +)) = e^{\downarrow}_{(m,t)}$ *and W maps onto edges* E_{WE}, *i.e.,* $\mu((m+1, t, +)) = e^{-}_{(m,t)}$.

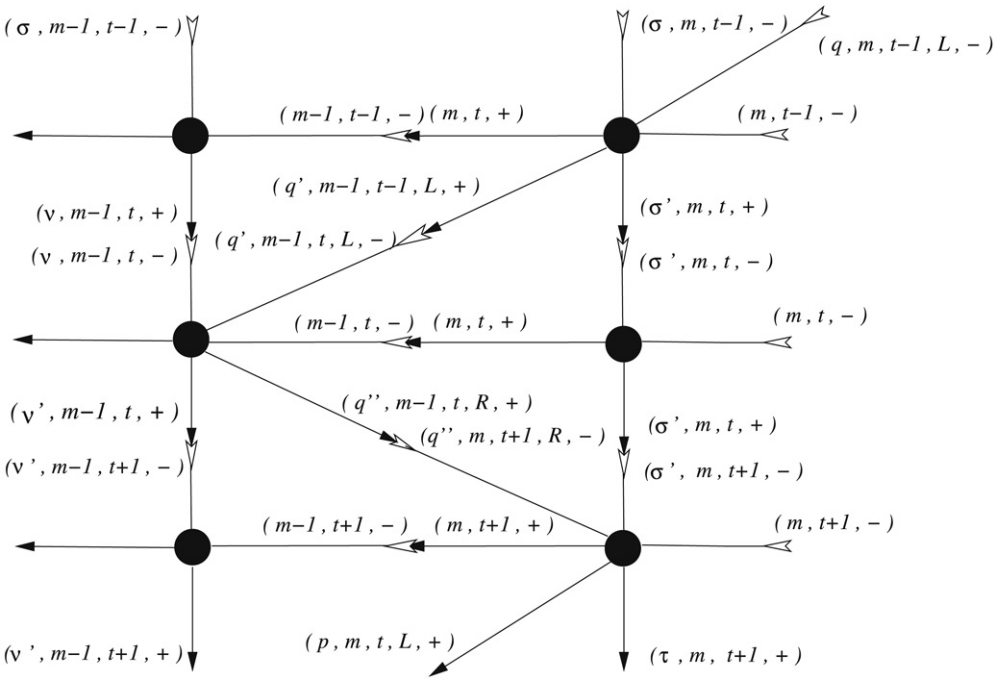

Fig. 8. Example of tiles simulating the transitions $\delta(q, \sigma) = (q', \sigma', L)$ when the tape head is at location m at time $t-1$, followed by $\delta(q', \nu) = (q'', \nu', R)$ at location $m-1$ at time t, and followed by $\delta(q'', \sigma') = (p, \tau, L)$ at location m at time $t+1$.

Then (G_{grid}, μ) *is an indexing system for* $\mathbf{P} = (\mathcal{M}, \mathbf{x}, l, r, t_{\max})$.

Proof. It is sufficient to observe that every complete complex in $\mathbf{P}$ maps homomorphically onto G_{grid}. Every complete complex has to have the side tiles (at locations l and r and at times 0 and $t_{\max}$) as every other type of tile has arms extending to all four directions. The side tiles impose that tiles representing locations m and time t for all t and m be included in the complex. Therefore, each complete complex contains a tile that can map onto (m, t) for every m and t. By construction, such a map extends to the required homomorphism.

We can now simulate Turing machine computations.

Theorem 6.1. *Given a non-deterministic Turing machine* $\mathcal{M}$*, time bound* $t_{\max}$*, and input* $\mathbf{x}$*, where* $|\mathbf{x}| \leq t_{\max} + 1$*, there is a computation of* $\mathcal{M}$ *accepting* $\mathbf{x}$ *within* $t_{\max}$ *steps if and only if the pot type* $(\mathcal{M}, \mathbf{x}, -t_{\max}, t_{\max}, t_{\max})$ *admits a complete complex of exactly* $2t_{\max}^2 + 3t_{\max} + 1$ *tiles.*

Moreover, if there is no nondeterministic computation of M accepting $\mathbf{x}$ *within* $t_{\max}$ *steps, then the pot* $(M, \mathbf{x}, -t_{\max}, t_{\max}, t_{\max})$ *admits no complete complexes of fewer than* $4t_{\max}^2 + 6t_{\max} + 2$ *tiles.*

Thus, the problem "does $\mathcal{M}$ accept x within $t_{\max}$ steps?" is satisfiable within bounds.

Proof. By the construction of the pot type $\mathbf{P} = (\mathcal{M}, \mathbf{x}, -t_{\max}, t_{\max}, t_{\max})$ if $\mathcal{M}$ halts within $t_{\max}$ steps, then there is a complete complex simulating a square grid of length $2t_{\max} + 1$ and width $t_{\max} + 1$. Each row of tiles in the complex represents a configuration of the Turing machine at a given time step.

Conversely, if there is a complete complex of exactly $2t_{\max}^2 + 3t_{\max} + 1$ tiles, then as its N–S and E–W edges form a grid-like index graph, it has to represent a simulation of the Turing machine computation which halts. For any m, t, $-t_{\max} \leq m \leq t_{\max}$ and $0 \leq t \leq t_{\max}$, the complete complex must have a tile that maps on a vertex in G_{grid} labeled m, t. We can "arrive" to this tile (corresponding to the vertex labeled m, t) by choosing an arbitrary tile in the complex (that maps to a vertex in G_{grid} to a vertex labeled, say, m_1, t_1). Going down E or W port connections to "arrive at a" tile (corresponding to vertex in G_{grid} labeled m, t_1, then down N or S connections to get to a tile corresponding to a vertex labeled m, t). As the size of the complete complex is within the bound $(2t_{\max} + 1)(t_{\max} + 1)$, the indexing system allows exactly one tile in the complex with label m, t for each m, t. Therefore, none of the boundary tiles containing label $t_{\max}$ has SW or SE ports, i.e., the complex encodes an accepting computation within $t_{\max}$ steps.

Note that the conditions of Lemma 5.1 are satisfied with the $(2t_{\max} + 1) \times (t_{\max} + 1)$ grid G_{grid} being the graph providing an indexing system for $\mathbf{P}$ (Lemma 6.1). Each tile has at least one port from H^* and μ is injective. If there is no computation of $\mathcal{M}$ accepting $\mathbf{s}$, within $t_{\max}$ steps, then the pot type admits no complete complex of less than $4t_{\max}^2 + 6t_{\max} + 2 = 2(2t_{\max}^2 + 3t_{\max} + 1)$ tiles.

Definition 15. A language $L \subseteq \Sigma^*$ is said to be *flexible tile assembly f-recognizable* (FTA(f)-recognizable) if there exists a DTIME(f) Turing machine which, given $u \in \Sigma^*$ outputs a (code for a) pot type $\mathbf{P}$, as well as a deranging involution θ and a positive integer $b < f(|u|)$ such that $u \in L$ if and only if $\mathbf{P}$ admits a complete complex of at most b tiles.

Given a class $\mathcal{F}$ of functions $f\colon \mathbb{N} \to \mathbb{N}$, let $\mathrm{FTA}(\mathcal{F}) = \bigcup_{f\in\mathcal{F}} \mathrm{FTA}(f)$.

The conclusion of this subsection is:

Corollary 6.1. *If $\mathcal{F}$ is polynomial closed, then* $\mathrm{NTIME}(\mathcal{F}) \subseteq \mathrm{FTA}(\mathcal{F})$.

Proof. If a language L is NTIME($\mathcal{F}$)-computable, then it is recognized by some nondeterministic Turing machine $\mathcal{M}$ of time bound function $f \in \mathcal{F}$. For every input string u with $|u| = n$, let $b = 2f(n)^2 + 3f(n) + 1$. By Theorem 6.1, u is accepted if and only if the simulation pot $\mathbf{P}_{\mathcal{M}}(u)$ corresponding to $\mathcal{M}$ with input u admits a complete complex of b tiles.

6.2. Computing within NPTIME

We now go in the reverse direction.

Theorem 6.2. *If $\mathcal{F}$ is polynomial closed, then* $\mathrm{FTA}(\mathcal{F}) \subseteq \mathrm{NTIME}(\mathcal{F})$.

Proof. For any language $L \in \mathrm{FTP}(f)$, $f \in \mathcal{F}$, the nondeterministic Turing Machine $\mathcal{M}$ recognizing L is obtained as follows. Given an input string u and letting $n = |u|$, $\mathcal{M}$ is to nondeterministically determine within time $cf(n)^k$ (for some fixed c, k) whether $\mathbf{P} = \mathbf{P}_L(u)$ admits a complete complex of at most $b = b_L(n)$ tiles. To do this, $\mathcal{M}$ (deterministically) composes a system of linear inequalities and equations that is soluble if and only if $\mathbf{P}$ admits a complete complex of at most b tiles, and then (nondeterministically) produce a solution to the system if one exists, and finally (deterministically) verifies that the alleged solution is in fact a solution to the system.

First, $\mathcal{M}$ enumerates a system of linear equations and inequalities as follows. $\mathcal{M}$ starts with a single inequality saying that there are at most b tiles in the assembled complex, so using $m_{\mathbf{t}}$ as the variable indicating the number of tiles of type $\mathbf{t} \in \mathbf{P}$ in the assembled complex, $\mathcal{M}$ writes

$$\sum_{\mathbf{t}\in\mathbf{P}} m_{\mathbf{t}} \leq b.$$

As each complex in the output of $\mathcal{M}$ has no free ports, for each $\mathbf{h} \in H$, the number of ports of type $\mathbf{h}$ in the complex is the same as the number of ports of type $\theta(\mathbf{h})$. Hence, for each $\mathbf{h} \in H$, $\mathcal{M}$ writes

$$\sum_{\mathbf{t}\in\mathbf{P}} m_{\mathbf{t}}\mathbf{t}(\mathbf{h}) - \sum_{\mathbf{t}\in\mathbf{P}} m_{\mathbf{t}}\mathbf{t}(\theta(\mathbf{h})) = 0.$$

Note that for each $\mathbf{h} \in H$ and $\mathbf{t} \in \mathbf{P}$, $\mathbf{t}(\mathbf{h})$ is fixed and given by the code for $\mathbf{P}$, so the expressions $\mathbf{t}(\mathbf{h}) - \mathbf{t}(\theta(\mathbf{h}))$ are the constant coefficients of the linear equation. (In addition, for each $\mathbf{t}$, we have the inequality $m_{\mathbf{t}} \geq 0$.)

As the number of tile types and port types are both bounded above by $f(n)$, each of the above equations can be composed on a work tape within time $O((f(n))^2 \log f(n))$ (one must go through the entire description of $\mathbf{P}$, and for each of the, at most $f(n)$ tile types, one must use the appropriate selections of the at most $f(n)$ port types, writing each of them down within time $\log f(n)$). As there are at most $O(f(n))$ equations, the entire system can be written down in time at most $O(f(n)^3 \log f(n))$.

Second, $\mathcal{M}$ nondeterministically guesses $|\mathbf{P}|$ values for the variables $m_{\mathbf{t}} \leq b$, writing each down within time $\log b \leq \log f(n)$, taking a total of at most $O(f(n) \log f(n))$ steps.

Third, $\mathcal{M}$ deterministically goes through the $|H|$ equations, in each one replacing variables with the guessed values, all within time $O(f(n)^2 \log f(n))$ (remembering that it may take as many as $O(f(n) \log f(n))$ steps to locate a single value on the list of guessed values). Then for each equation, $\mathcal{M}$ computes and adds all the terms within time $O(f(n)^2(\log f(n))^2)$ (the multiplications of numbers of at most $f(n)$ may take quadratic time with respect to the space $\log f(n)$ that they occupy), verifying or impeaching the equation or inequality. As there are at most $O(f(n))$ equations, all this takes $O(f(n)^3(\log f(n))^2)$ steps.

Thus $\mathcal{M}$ operates within time

$$\begin{aligned} O(f(n)^3 \log f(n)) + O(f(n) \log f(n)) + O(f(n)^3(\log f(n))^2) &= O(f(n)^3(\log f(n))^2) \\ &\leq O(f(n)^4), \end{aligned}$$

so for some function f' with $f'(n) = cf(n)^4$ for some constant c, $L \in \mathrm{NTIME}(f')$. As $\mathcal{F}$ is polynomial closed, $f' \in \mathcal{F}$, so $L \in \mathrm{NTIME}(\mathcal{F})$.

We pull Corollary 6.1 and Theorem 6.2 together to get the main result:

Corollary 6.2. *If $\mathcal{F}$ is polynomial closed, then* $\mathrm{NTIME}(\mathcal{F}) = \mathrm{FTA}(\mathcal{F})$.

In particular, we get the result alluded to at the end of Section 4: the NPTIME-recognizable languages are precisely the FTA(polynomial)-recognizable languages. But the polynomials are merely the smallest polynomial closed class of functions. We have a hierarchy of results for larger classes $\mathcal{F}$. For example, for each real $a > 0$, if $\mathcal{F}_a$ was the class of functions bounded above by functions $n \mapsto n^{n^{a+o(1)}}$, then $\mathcal{F}_a$ would be a polynomial closed class of functions, and hence $\mathrm{FTA}(\mathcal{F}_a) = \mathrm{NPTIME}(\mathcal{F}_a)$. And letting $\mathrm{EXP} = \bigcup_a \mathcal{F}_a$, we get FTA(EXP) = NEXPTIME. And so on up.

7. Concluding remarks

3SAT and several graph theoretic problems can be solved by the Flexible Tile Assembly model in a rather straightforward way, with much fewer tile types than the FTA simulations of appropriate Turing Machines. In this paper, we simulated Turing Machines in order to obtain representation theorems. However, the real power of this assembly model is that it utilizes a graphical structure naturally embeddable in three dimensional Euclidean space, and thus readily encodes an array of combinatorial problems. This suggests two points. First, this model of computation is different from many extant models in that the structure of the computer itself changes during the computation. In Turing Machine and Register Machine computations, energy (which we can treat as information) is shuttled about an essentially static machine. In Flexible Tile Assembly, as with all assembly process computations, the physical structure of the machine itself changes. This leads to our second point. Since the computations are fundamentally different, it would seem likely that both the computational and descriptive complexity of queries would be different. A finer analysis of space complexity for Flexible Tile Assembly may produce a quite different hierarchy than the (apparent) hierarchies of Turing Machines, especially at the lower levels. On the other hand, the fact that NPTIME, NEXPTIME (and NEXP^2TIME and ...) are naturally represented in both models suggests that the higher levels are more robust.

Acknowledgment

This work has been partially supported by the grant CCF-0523928 and CCF-0726396 from the National Science Foundation, USA.

References

[1] L. Adleman, Molecular computation of solutions of combinatorial problems, Science 266 (1994) 1021–1024.
[2] L.M. Adleman, Q. Cheng, A. Goel, M.-D. Huang, D. Kempe, P. Moisset de Espanés, P.W.K. Rothemund, Combinatorial optimization problems in self-assembly, in: STOC'02 Proceedings, Montreal Quebec, Canada, 2002.
[3] L.M. Adleman, J. Kari, L. Kari, D. Reishus, On the decidability of self-assembly of infinite ribbons, in: Proceedings of FOCS 2002, IEEE Symposium on Foundations of Computer Science, Washington, 2002, pp. 530–537.
[4] S. Angelov, S. Khanna, M. Visontai, On the complexity of graph self-assembly in accretive systems, Natural Computing (October) (2007).
[5] R.D. Barish, P.W.K. Rothemund, E. Winfree, Two computational primitives for algorithmic self-assembly: Copying and counting, Nanoletters 5 (12) (2005) 2586–2592.
[6] J.H. Chen, N.C. Seeman, Synthesis from DNA of a molecule with the connectivity of a cube, Nature 350 (1991) 631–633.
[7] S. Cook, The complexity of theorem proving procedures, in: Proc. 3rd. ACM Ann. Symp. Theory of Comput., 1971, pp. 151–158.
[8] R. Fagin, Generalized first-order spectra and polynomial-time recognizable sets, in: R. Karp (Ed.), Complexity of Computation, in: SIAM–AMS Proceedings, vol. 7, 1974, pp. 43–73.
[9] R.P. Goodman, I.A. Schaap, C.F. Tardin, C.M. Erben, R.M. Berry, C.F. Schmidt, A.J. Turberfield, Rapid chiral assembly of rigid DNA building blocks for molecular nanofabrication, Science 310 (5754) (2005) 1661–1665.
[10] D. Faulhammer, A.R. Curkas, R.J. Lipton, L.F. Landweber, Molecular computation: RNA solution to chess problems, Proceedings of the National Academy of Sciences 97 (2000) 1385–1389.
[11] T.J. Fu, N.C. Seeman, DNA double crossover structures, Biochemistry 32 (1993) 3211–3220.
[12] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness, W. H. Freeman, 1979.
[13] T. Head, et al., Computing with DNA by operating on plasmids, BioSystems 57 (2000) 87–93.
[14] N. Jonoska, P. Sa-Ardyen, N.C. Seeman, Computation by self-assembly of DNA graphs, Genetic Programming and Evolvable Machines 4 (2003) 123–137.
[15] N. Jonoska, S. Karl, M. Saito, Three dimensional DNA structures in computing, BioSystems 52 (1999) 143–153.
[16] N. Jonoska, S. Karl, M. Saito, Creating 3-dimensional graph structures with DNA, in: H. Rubin, D. Wood (Eds.), DNA Based Computers III, in: AMS DIMACS Series, vol. 48, 1999, pp. 123–136.
[17] N. Jonoska, G. McColm, A computational model for self-assembling flexible tiles, in: Cristian S. Calude, Michael J. Dinneen, Gheorghe Paun, Mario J. Pérez-Jiménez, Grzegorz Rozenberg (Eds.), Proceedings of the 4th International Conference on Unconventional Computation (Sevilla, Spain, October 2005), in: LNCS, vol. 3699, Springer, 2005, pp. 142–156.
[18] N. Jonoska, G. McColm, Flexible versus rigid tile assembly, in: Gh. Paun, et al. (Eds.), Unconventional Computers, in: LNCS, vol. 4135, Springer, 2006, pp. 139–151.
[19] N. Jonoska, G. McColm, A. Staninska, Expectation and variance of self-assembled graph structures, in: Proceedings of 11th International Meeting on DNA Computing, London, Ontario, Canada, June 6–9, 2005, pp. 49–58.
[20] N. Jonoska, G. McColm, A. Staninska, Spectrum of a pot for DNA complexes in *DNA Computing 12*, in: C. Mao, T. Yokomori (Eds.), in: LNCS, vol. 4287, Springer, 2006, pp. 83–94.
[21] M-Y. Kao, V. Ramachandran, DNA self-assembly for constructing 3D boxes, in: Algorithms and Computation: ISAAC 2001 Proceedings, in: LNCS, vol. 2223, Springer, 2001, pp. 429–440.
[22] T.H. LaBean, H. Yan, J. Kopatsch, F. Liu, E. Winfree, J.H. Reif, N.C. Seeman, The construction, analysis, ligation and self-assembly of DNA triple crossover complexes, Journal of American Chemical Society 122 (2000) 1848–1860.
[23] C. Mao, T.H. LaBean, J.H. Reif, N.C. Seeman, Logical computation using algorithmic self-assembly of DNA triple-crossover molecules, Nature 407 (2000) 493–496.
[24] J.H. Reif, S. Sahu, P. Yin, Complexity of graph self-assembly in accretive systems and self-destructive systems, in: Proceedings of 11th International Meeting on DNA Computing, London, Ontario, Canada, June 6-9, 2005, pp. 101–112.
[25] P.W. Rothemund, Folding DNA to create nanoscale shapes and patterns, Nature 440 (7082) (2006) 297–302.
[26] P.W.K. Rothemund, E. Winfree, The program-size complexity of self-assembled squares, in: Proceedings of 33rd ACM meeting STOC 2001, Portland, Oregon, May 21–23, 2001, pp. 459–468.
[27] P. Rothemund, N. Papadakis, E. Winfree, Algorithmic self-assembly of DNA Sierpinski triangles, PLoS Biology 2 (12) (2004) e424.
[28] P. Sa-Ardyen, N. Jonoska, N. Seeman, Self-assembly of graphs represented by DNA helix axis topology, Journal of American Chemical Society 126 (21) (2004) 6648–6657.
[29] N.C. Seeman, DNA junctions and lattices, Journal of Theoretical Biology 99 (1982) 237–247.
[30] N.C. Seeman, DNA nicks and nodes and nanotechnology, NanoLetters 1 (2001) 22–26.
[31] W.M. Shihm, J.D. Quispe, G.F. Joyce, A 1.7-kilobase single-stranded DNA folds into a nano-scale octahedron, Nature 427 (Feb.) (2004) 618–621.
[32] D. Soloveichik, E. Winfree, Complexity of self-assembled shapes, preprint at http://arxiv.org/abs/cs.CC/0412096.

[33] L. Stockmeyer, The polynomial-time hierarchy, Theoretical Computer Science 3 (1974) 1–22.
[34] Y. Wang, J.E. Mueller, B. Kemper, N.C. Seeman, The assembly and characterization of 5-arm and 6-arm DNA junctions, Biochemistry 30 (1991) 5667–5674.
[35] X. Wang, N.C. Seeman, Assembly and characterization of 8-arm and 12-arm DNA branched junctions, Journal of American Chemical Society (Jun) (2007).
[36] E. Winfree, X. Yang, N.C. Seeman, Universal computation via self-assembly of DNA: Some theory and experiments, in: L. Landweber, E. Baum (Eds.), DNA Based Computers II, in: AMS DIMACS series, vol. 44, 1998, pp. 191–214.
[37] E. Winfree, F. Liu, L.A. Wenzler, N.C. Seeman, Design and self-assembly of two-dimensional DNA crystals, Nature 394 (1998) 539–544.
[38] G. Wu, N. Jonoska, N.C. Seeman, Self-assembly of a DNA nano-object demonstrates natural computation (submitted for publication).
[39] Y. Zhang, N.C. Seeman, The construction of a DNA truncated octahedron, Journal of American Chemical Society 160 (1994) 1661–1669.